



Politechnika Śląska

PROJEKT INŻYNIERSKI

Analiza błędów pomiaru położenia platformy mobilnej

Daniel CHYDZIŃSKI

Nr albumu: 296781

Kierunek: Automatyka i Robotyka

Specjalność: Automatyka Procesowa

PROWADZĄCY PRACĘ

dr Aleksander Staszulonek

KATEDRA Automatyki i Robotyki

Wydział Automatyki, Elektroniki i Informatyki

Gliwice 2024

Tytuł pracy

Analiza błędów pomiaru położenia platformy mobilnej

Streszczenie

Projekt, wykonanie i testy platformy mobilnej opartej na mikroprocesorze, silnikach prądu stałego z enkoderami i regulatorach PID.

Słowa kluczowe

Silnik, druk 3D, regulator PID, mikroprocesor

Thesis title

Mobile platform position errors analysis

Abstract

Design, development and testing of a mobile platform based on a microprocessor, direct current motors with encoders and PID controllers.

Key words

Motor, 3D printing, PID controller, microprocessor

Spis treści

1 Wstęp	1
2 Analiza tematu	3
2.1 Detekcja położenia wału silnika	3
2.2 Regulacja sygnału sterującego	6
3 Założenia i narzędzia	9
3.1 Założenia	9
3.2 Narzędzia	11
4 Projekt i wykonanie	13
4.1 Pojazd	13
4.2 Oprogramowanie pomocnicze	27
5 Eksperyment	29
6 Podsumowanie i wnioski	31
Bibliografia	34
Spis skrótów i symboli	37
Źródła	39
Lista dodatkowych plików, uzupełniających tekst pracy	43
Spis rysunków	45
Spis tabel	47

Rozdział 1

Wstęp

Robotyka to obszar badawczy i techniczny poświęcony teorii, konstrukcji oraz praktycznym zastosowaniom robotów. Elementami wykonawczymi układów zrobotyzowanych są najczęściej silniki lub siłowniki, te drugie nierzadko napędzane wewnętrznie silnikami. Rodzajem maszyny zamieniającej jeden rodzaj energii — w robotyce najczęściej elektryczną — na energię mechaniczną, czego celem jest wprawienie w ruch elementów ruchomych.

W przypadku prostych układów (Definicja 1), takich jak podajnik taśmowy napędzany pojedynczym silnikiem, dokładność (Definicja 3) sterowania nie ma wysokiego priorytetu. Najważniejsze jest, by element znajdujący się na taśmie przejechał z punktu A do punktu B z pewną prędkością, a jego położeniem zajmą się inne czujniki. Jednak gdy silnik napędza ramię robota, pojazd lub drona, ważne jest, by utrzymywał stałą prędkość i/lub wykonywał określoną ilość obrotów.

Definicja 1 (Układ sterowania). *Układ służący do kontroli pożądanego urządzenia przy pomocy wybranego zasobu narzędzi, w tym zamkniętej pętli z ujemnym sprzężeniem zwrotnym (Definicja 2).*

Definicja 2 (Pętla). *Typ struktury układu umożliwiający mu reakcję na informację zwrotną o jego stanie (sprzężenie zwrotne) pochodzącej z czujników.*

Definicja 3 (Dokładność). *Stopień zgodności pomiędzy rzeczywistymi wartościami a wartościami określonymi lub mierzonymi.*

Z tego powodu, jednym z wyzwań, z jakimi mierzyli się pionierzy automatyczno-robotyczni jest dokładne sterowanie tworzonymi przez siebie układami. Jest to kwestia o tyle istotna, że gdy odpowiedź układu odbiega — nawet w niewielkim stopniu — od wartości zadanej, staje się on znaczowo trudniejszy w użytkowaniu (sterowaniu), a w skrajnych przypadkach bezużyteczny. W przypadku ramienia robota, brak dokładności sterowania doprowadzi układ do osiągnięcia innej lokalizacji niż pożądana. W przypadku drona, prawdopodobnie całkowicie uniemożliwi lot z powodu nierówności sił nośnych. Zaś w przypadku po-

jazdów autonomicznych, przejechanie odległości innej niż zadana przez operatora. Jako rozwiązanie tego problemu, powstał poddział robotyki zwany odometrią (Definicja 4).

Definicja 4 (Odometria). *Dział nauk technicznych na pograniczu robotyki i miernictwa, zajmujący się użyciem różnego rodzaju czujników w celu oszacowania położenia ruchomego układu względem pozycji startowej w przestrzeni fizycznej.*

Współcześni automatycy-robotycy będący na początku swojej ścieżki edukacji/kariery, lub zajmujący się robotyką jedynie hobbystycznie, również mierzą się przedżej czy później z problemem dokładności sterowania układu. W obecnych czasach istnieje wiele sposobów rozwiązania go.

Celem niniejszej pracy inżynierskiej jest zaprojektowanie i wykonanie fizycznego modelu platformy mobilnej (zwanej dalej pojazdem) oraz aplikacji do jej kontroli. Następnie zastosowanie i przetestowanie eksperymentalne wybranej metody zwiększenia dokładności układu z użyciem ujemnego sprzężenia zwrotnego. Na końcu otrzymane wyniki zostaną przeanalizowane pod kątem użyteczności i skuteczności wybranej metody w warunkach rzeczywistych.

Praca podzielona została na następujące rozdziały[18]:

1. Wstęp — wprowadzenie do tematu, krótkie omówienie istoty problemu, zakres pracy, opis rozdziałów.
2. Analiza tematu — omówienie tematu w kontekście aktualnego stanu wiedzy o poruszonym problemie (ang. *state of art*), oraz szczegółowe jego sformułowanie.
3. Założenia i narzędzia — opis wymagań postawionych przy tworzeniu projektu wraz z uzasadnieniem wyboru, a także lista użytych narzędzi.
4. Projekt i wykonanie — szczegółowe omówienie sposobu wykonania modelu pojazdu, układu elektronicznego, aplikacji, kodu mikroprocesora i narzędzi pomocniczych. Każdemu elementowi odpowiada podrozdział.
5. Eksperyment i weryfikacja — zastosowane metody badawcze i wykonane eksperymenty.
6. Podsumowanie i wnioski — skomentowanie uzyskanych wyników pod kątem osiągnięcia założonych celów, analiza i dalsze kroki.

Rozdział 2

Analiza tematu

Najistotniejszą składową projektu jest synchronizacja prędkości silników pojazdu. Dlatego analizując poruszoną tematykę, zdecydowano się na pominięcie szczegółowych opisów technologii pobocznych, takich jak druk 3D (ang. *3 dimensions*). W bieżącym rozdziale opisano przeprowadzono rozważania jedynie na temat technologii pomiaru położenia wałów silników, oraz metod wyznaczania sygnałów sterujących.

2.1 Detekcja położenia wału silnika

Problem synchronizowania silników elektrycznych znany jest w dziedzinie automatyki od dziesiątek lat. Początki badania silników sięgają XIX wieku, kiedy to Michael Faraday oraz inni naukowcy eksperymentowali ze wykorzystaniem elektromagnetyzmu[3]. Pierwsze silniki elektryczne były prymitywne i nie miały zaawansowanych metod sterowania. Wczesne próby pozycjonowania opierały się głównie na prostych mechanizmach, takich jak przekładnie i sprzęgła.

W miarę postępu technologicznego, szczególnie w XX wieku, rozwijano bardziej zaawansowane metody pozycjonowania. Pojawiły się pierwsze systemy sterowania, wykorzystujące technologię zwrotną informacji, mającą na celu monitorowanie i regulację położenia wałów silników. Jednak precyzja tych rozwiązań była ograniczona, a dokładność pozycjonowania nie zawsze spełniała wymagania coraz bardziej zaawansowanych zastosowań.

Dopiero wprowadzenie enkoderów (Definicja 5) elektronicznych w latach 60. XX wieku[17] stało się przełomem.

Definicja 5 (Enkoder obrotowy). *Urządzenie, generujące sygnały elektryczne odpowiadające ruchowi obrotowemu wału silnika celem określenia jego pozycji.*

Początkowo enkodery były oparte na szczotkach stykających się z dyskiem zawierającym serię odpowiednio zakodowanych pierścieni koncentrycznych (Rysunek 2.1), wy pełnionych otworami o odpowiedniej długości[7]. Są one tanie w produkcji, jednak mają

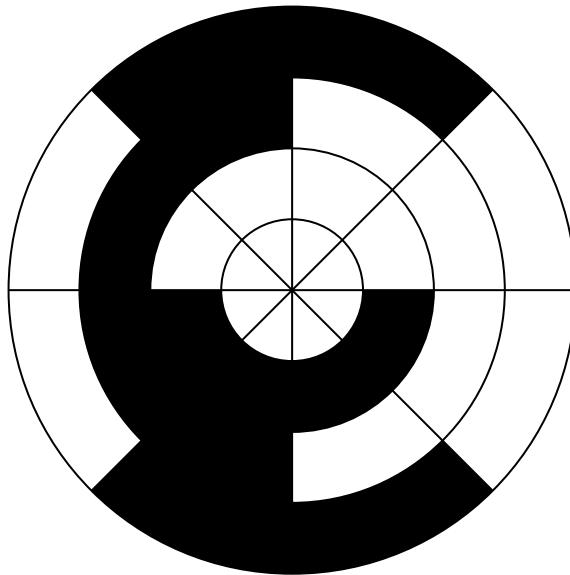
swoje ograniczenia związane ze zużyciem mechanicznym elementów stykowych, niską maksymalną dozwoloną prędkością silnika i wymaganiami konserwacji. Ten typ enkoderów spotykany jest do dziś, na przykład w multimetrach cyfrowych.

Rozwój technologii przyniósł enkodery optyczne, wykorzystujące diody LED i fotodetektory. Później pojawiły się enkodery magnetyczne. To właśnie one — enkodery optyczne i magnetyczne — są do dnia dzisiejszego najczęściej spotykane i oferują najwyższą dokładność sterowania przy niskich kosztach i niewielkim stopniu skomplikowania. To właśnie na nich skupiono się w dalszej części pracy.

Enkodery można podzielić ze względu na[7]:

- Metodę używaną do odczytania pozycji: kontaktowe i bezkontaktowe.
- Rodzaj sygnału wyjściowego: pozycja absolutna lub szereg inkrementujących/dekrementujących wartości.
- Zjawisko fizyczne wykorzystane do przesłania sygnału pozycyjnego: przewodzenie elektryczne, magnetyzm, zjawiska optyczne lub pojemnościowe.

Najważniejszy jest podział ze względu na rodzaj sygnału wyjściowego. Mimo, że zarówno enkodery absolutne jak i inkrementalne posiadają dyski kodujące, różnią się one działaniem. Enkodery absolutne jako sygnał wyjściowy podają precyzyjną pozycję wału silnika, najczęściej zakodowaną w słowie bitowym. Przykładowy wygląd dysku kodującego widoczny jest na Rysunku 2.1. Istotną cechą tego rodzaju enkoderów jest możliwość określenia pozycji nawet po utracie zasilania.

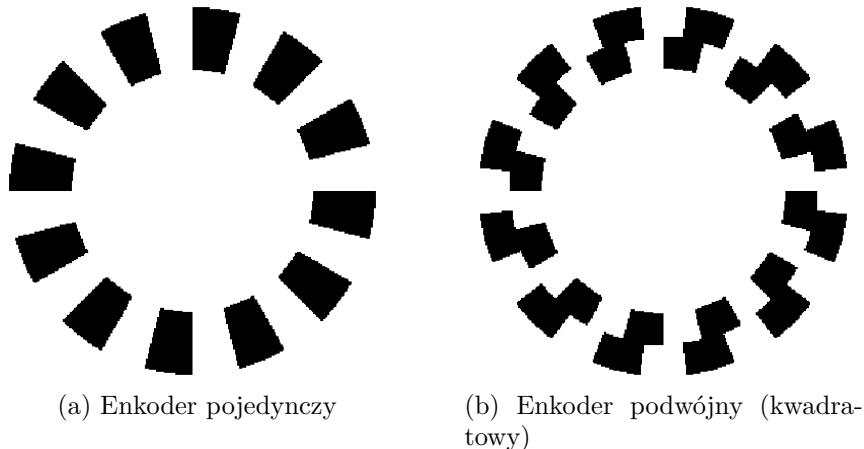


Rysunek 2.1: Poglądowy schemat dysku enkodera absolutnego z 3-bitowym kodem Graya (Bibliografia: [4])

Enkodery inkrementalne u podstaw działają w ten sam sposób, tzn. opierają się na dyskach kodujących, z tą różnicą, że nie są w stanie podać dokładnej wartości położenia. Zamiast tego, podają na wyjściu odpowiedni impuls przy obrocie w danym kierunku. Następnie w oprogramowaniu impulsy te są zliczane w celu oszacowania aktualnej pozycji

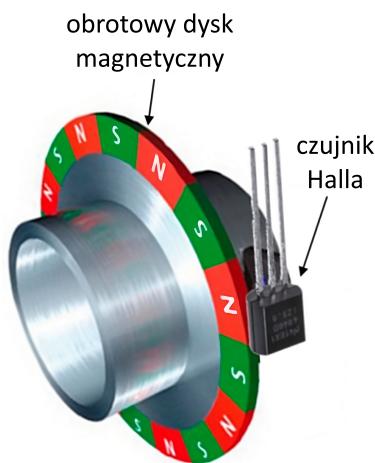
względem pozycji startowej. Ze względu na wyzerowanie liczby impulsów przy utracie zasilania, ten typ enkodera nie jest w stanie podać dokładnej pozycji w przypadku utraty zasilania.

Istotny jest również podział enkoderów ze względu na wykorzystywane zjawisko fizyczne. Dwa główne typy to enkodery optyczne oraz magnetyczne. Oba rodzaje występują zarówno w wariantie pojedynczym (Rysunek 2.2a) jak i podwójnym (Rysunek 2.2b). W przypadku enkoderów optycznych, kolorowi białemu odpowiada szczelina, zaś kolorowi czarnemu blokada. W przypadku enkoderów magnetycznych, kolorom odpowiadają biegunki S i N.



Rysunek 2.2: Poglądowe schematy dysku enkodera inkrementalnego (Bibliografia: [6])

Jako że w projekcie wykorzystane zostały enkodery magnetyczne, zasada ich działania została opisana, zaś enkodery optyczne pominięto. W enkoderze magnetycznym umieszcza się magnesy na obracającej się lub przemieszczającej się części obiektu — czyli dysku — a czujniki magnetyczne — najczęściej czujniki Halla (Definicja 6) — znajdujące się na stałej części enkodera rejestrują zmiany pola magnetycznego (Rysunek 2.3).



Rysunek 2.3: Zobrazowanie zasady działania enkodera magnetycznego dla tarczy z jednym рядом (Bibliografia: [12])

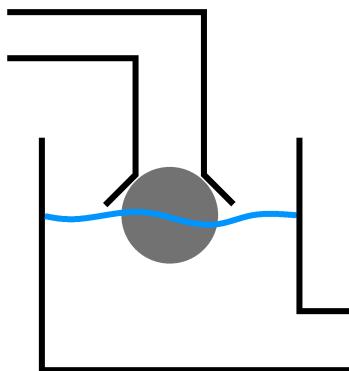
Definicja 6 (Czujnik Halla). *Czujnik pola magnetycznego i prądu, wykorzystujący zjawisko Halla.*

Te zmiany są następnie przetwarzane na sygnały elektryczne, które można interpretować jako informacje o kącie obrotu lub przemieszczeniu. Enkodery magnetyczne charakteryzują się wysoką dokładnością pomiaru, odpornością na wibracje i brakiem kontaktu mechanicznego, dzięki czemu są odporne na zabrudzenia i czynniki zewnętrzne takie jak kurz. Nie ma więc potrzeby osadzania ich w zamkniętej przestrzeni, jak to ma miejsce z czujnikami optycznymi.

2.2 Regulacja sygnału sterującego

Oprócz urządzenia (czujnika) dostarczającego sygnał informujący o fizycznym położeniu lub przemieszczeniu wału silnika, konieczne jest również zastosowanie metody wyznaczenia sygnału sterującego u .

Trudno jest określić datę powstania pierwszych regulatorów. Pierwotnie nie były one tworzone z myślą o wyznaczaniu sygnału sterującego, lecz zwyczajnie jako część urządzeń. Przykładem jednego z pierwszych regulatorów może być maszyna Ktesibiosa — wynaleziona w III wieku p.n.e.[2] — w której rolę regulatora pełnił pływak dławiaczy wypływającą ze źródła wodę (Rysunek 2.4). Jest to prawdopodobnie pierwszy na świecie przykład regulatora proporcjonalnego.



Rysunek 2.4: Zobrazowanie zasady działania regulatora Ktesibiosa

Na przestrzeni kolejnych tysięcy lat w technologii regulatorów nie dokonał się w zasadzie żaden znaczący postęp. Dopiero w XIX i XX wieku rozpoczęły się badania nad teorią sterowania, w których udział miało wielu naukowców z całego świata. Lata prac doprowadziły do formalnego opracowania w roku 1922 regulatora PID (ang. *Proportional–Integral–Derivative*) (Definicja 7) przez rosyjskiego naukowca, Nicolasa Minorsky'ego[5].

Definicja 7 (Regulator PID (Proporcjonalno-Integrująco-Różniczkujący)). *Rodzaj regulatora, który składa się z trzech elementów: proporcjonalnego (P), który reaguje na bieżącą*

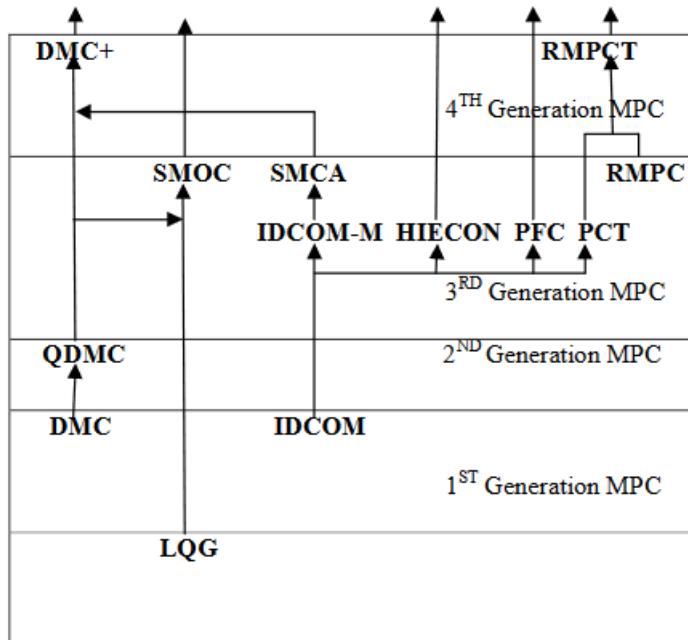
wartość błędu (*Definicja 8*) proporcjonalnie do niego; całkującego (*I*), który integruje bieżący błąd w czasie i reaguje na jego sumę; oraz różniczkującego (*D*), które reaguje na szybkość zmiany błędu.

Definicja 8 (Błąd; uchyb). *W układach automatyki jest to różnica między wartością zadaną a zmierzoną wartością rzeczywistą.*

Jego popularyzacja nastąpiła w latach '50 XX wieku, gdy układy elektroniczne stały się tańsze, dotepniejsze i bardziej niezawodne. Dziś jest powszechnie stosowany w automatyce, robotyce, elektronice i wielu innych dziedzinach inżynierii. Regulator PID jest stosowany najczęściej ze względu na prostotę, bardzo niski koszt implementacji i wystarczające działanie w większości przypadków.

Drugim często stosowanym regulatorem jest regulator predykcyjny MPC (ang. *Model Predictive Control*). W przeciwieństwie do tradycyjnych regulatorów które reagują na sprzężenie zwrotne z wyjścia układu, regulator predykcyjny działa z wyprzedzeniem, zanim zdąży nastąpić zmiana na wyjściu układu.

Pierwsze wzmianki o tym typie kontroli datowane są na wczesne lata '60 XX wieku i rozważania Rudolfa E. Kálmána na temat systemów liniowych. Od tego czasu, technologia regulatora predykcyjnego była rozwijana niezależnie przez wiele osób i instytucji. W późnych latach '70 XX wieku regulator ten można było już spotkać w zastosowaniach przemysłowych. W roku 1979 zaprezentowano generację pierwszą, zaś najnowszą, 4 generację stanowią w roku 1998[10]. Pełna genealogia algorytmów MPC przedstawiona została na Rysunku 2.5.



Rysunek 2.5: Genealogia algorytmów MPC (Bibliografia: [10])

Regulatory predykcyjne znajdują zastosowanie w układach o dużym opóźnieniu i wysokim rzędzie, gdzie regulatory PID są niewystarczające. Przykładem są sondy kosmiczne.

Ostatnim wymienionym sposobem sterowania jest sterowanie ślizgowe, znane również jako zmiennostrukturalne. Polega na zastosowaniu nieciągłego sygnału sterującego poprzez szybkie przełączanie sterowania, kiedy stan systemu osiąga tzw. powierzchnię ślizgową. Charakteryzuje się efektywnością zwłaszcza w warunkach niepewności i zmiennych warunków operacyjnych, a także radzi sobie z nieliniowościami i zapewnia szybką odpowiedź[1]. Wadami jest zjawisko drgań (ang. *chattering*) co negatywnie wpływa na elementy wykonawcze, oraz wysoki stopień skomplikowania wymagający dogłębnej analizy projektowanego systemu, co może stać w sprzeczności z założeniami projektowymi — jak w przypadku niniejszego projektu. Metoda została wynaleziona w latach '70 XX wieku przez rosyjskiego uczonego Vadima I. Utkina[9].

Rozdział 3

Założenia i narzędzia

Przed przystąpieniem do wykonania projektu, wskazane jest opisanie założeń projektowych i krótki spis narzędzi wykorzystanych do ich zrealizowania. Założenia podzielone zostały na kilka podsekcji, po jednej dla każdej części projektu. Wyjaśnienie poszczególnych części/modułów znajduje się w Rozdziale 4.

3.1 Założenia

Dla układu elektronicznego

- 2 silniki DC (ang. *Direct Current*)
- sterownik silników
- czujnik laserowy z przodu pojazdu
- enkodery magnetyczne do pomiaru położenia wałów silników
- sterowanie przy użyciu mikroprocesora
- LED (ang. *Light Emitting Diode*) sygnalizujący stan oprogramowania mikroprocesora
- głośnik sygnalizujący stan oprogramowania mikroprocesora
- bezpieczniki zabezpieczające układ elektroniczny
- przełączniki źródeł prądowych

Dla modelu pojazdu

- możliwa jazda do przodu, tyłu oraz skręcanie jak w samochodzie osobowym
- 4 koła, kilka zestawów rozmiarów dla różnorodności eksperymentalnej
- modułowość pozwalająca na modyfikację w razie potrzeby
- projekt wizualny podobny do samochodu osobowego
- obudowa wydrukowana na drukarce 3D

Dla oprogramowania mikroprocesora

- system FreeRTOS (ang. *Real Time Operating System*)
- wykonanie w języku C++
- klient i serwer UDP (ang. *User Datagram Protocol*)
- interpretacja danych z ramek pakietów UDP
- sterowanie możliwe w układzie otwartym lub zamkniętym
- wartość zadana odbierana z aplikacji mobilnej przez Wi-Fi (ang. *Wireless Fidelity*)
- rozpoczęcie i zakończenie pomiaru na komendę z aplikacji mobilnej
- awaryjne zakończenie pomiaru w przypadku rozłączenia Wi-Fi
- odczytywanie kierunku i położenia enkoderów
- synchronizacja silników
- obliczanie sygnałów sterujących silników regulatorami PID
- plik konfiguracyjny

Dla aplikacji mobilnej

- działanie na systemie Android
- prostota wykonania i użytkowania
- krótki czas tworzenia (ang. *development*)
- klient i serwer UDP
- interpretacja danych z ramek pakietów UDP
- zmienny docelowy adres IP (ang. *Internet Protocol*)
- parametryzacja regulatorów PID
- ustawianie wartości zadanej i prędkości

Dla skryptu odbierającego dane

- działanie na systemie Windows
- wykonanie w języku Python
- serwer UDP
- interpretacja danych z ramek pakietów UDP
- zapis danych do pliku .csv (ang. *Comma-Separated Values*)
- działanie w pętli; możliwość odbioru wielu pomiarów

Dla skryptu wizualizującego dane

- działanie na systemie windows
- odczytywanie danych z pliku .csv
- wizualizacja odczytanych danych (tworzenie wykresów)
- zapisywanie wykresów do pliku .eps (ang. *Encapsulated PostScript*)

3.2 Narzędzia

Narzędzia fizyczne

- Zestaw lutowniczy Lutowanie to proces łączenia elementów elektronicznych przez stopienie spoiwa lutowniczego na przylegających do siebie elementach metalowych, a następnie jego zastygnięcie. Do lutowania wykorzystane zostały następujące narzędzia:
 - lutownica transformatorowa 100 W typu B
 - topnik w żelu
 - plecionka do rozlutowywania
 - cyna lutownicza bezołowiowa z 3.8% srebra
 - alkohol izopropylowy 100%
 - zestaw przewodów typu prototypowego (ang. *jumper wires*)
 - folia aluminiowa

Lutownica jest, oprócz drukarki 3D, najważniejszym narzędziem użyтыm w projekcie.

- Czarna taśma izolacyjna o wielofunkcyjnym zastosowaniu — izolacja elementów elektrycznych, ale również czynnik pomocniczy, zwiększający tarcie i łączący luźne elementy.
- Pistolet do kleju na gorąco posłużył do przytwierdzania elementów w miejscu.
- Multimetr do pomiaru wielkości elektrycznych. W projekcie użyto modelu UNI-T M830BUZ z funkcją mierzenia napięcia, natężenia, rezystancji i ciągłości.
- Drukarka 3D (Rysunek 3.1), na której wydrukowano obudowę oraz koła pojazdu.



Rysunek 3.1: Drukarka 3D model Sovol SV06 (Bibliografia: [14])

Oprogramowanie

Visual Studio Code

Do napisania oprogramowania mikroprocesora oraz pracy inżynierskiej użyta została platforma Visual Studio Code, z rozszerzeniami LaTeX Workshop i PlatformIO IDE.

MATLAB

MATLABa użyto w celu wizualizacji danych otrzymanych z pojazdu.

GitHub

GitHub posłużył jako system kontroli wersji, zarówno do kodu mikroprocesora, jak i do skryptów pomocniczych i pracy inżynierskiej.

MIT App Inventor

MIT App Inventor wykorzystany został do stworzenia aplikacji na system Android.

Autodesk Inventor

W Inventorze zaprojektowano modele 3D pojazdu do druku 3D.

Paint.NET

Program do obróbki graficznej, utworzono w nim schemat układu elektronicznego.

Rozdział 4

Projekt i wykonanie

Projekt składa się z kilku współpracujących ze sobą modułów, posiadających odrębne role. Każdy z nich wykonano osobno, a następnie połączono — pośrednio lub bezpośrednio — w jeden, działający system.

Pierwszym elementem jest fizyczny pojazd, na który składa się układ elektroniczny i wydrukowany model 3D. Jest on elementem centralnym, wokół którego budowana jest reszta systemu.

Drugim elementem jest aplikacja mobilna na system Android. Jej zadaniem jest wysyłanie do pojazdu informacji o nowym pomiarze, jego rozpoczęcie oraz zakończenie.

Trzeci element to skrypt w języku Python. Odbiera on z pojazdu dane pomiarowe i zapisuje je do pliku .csv celem dalszej pracy na uzyskanych danych.

Ostatni element to skrypt w języku MATLAB. Odczytuje on zapisane poprzednio dane z pliku .csv i przetwarza je w celu prezentacji.

4.1 Pojazd

Projekt pojazdu zakłada 4 koła, z czego 2 przednie na wspólnej osi skrętnej, a 2 tylne na osi statycznej obrotowej zasilane silnikami DC. W pojeździe muszą znajdować się sloty na akumulatory zasilające, przestrzeń na układ elektroniczny z przewodami, oraz musi zostać zapewniony sposób stabilnego montażu silników, tak by zminimalizować wpływ zakłóceń pomiarowych spowodowanych przez drgania i przemieszczanie się osi w trakcie działania układu.

Druk 3D

Technologia druku 3D opiera się na nakładaniu na siebie kolejnych, cienkich warstw stopionego materiału. Jest to metoda addytywna obróbki materiału. Dwie główne technologie druku to FDM (ang. *Fused Deposition Modeling*) oraz DLP (ang. *Digital Light Processing*), w których stosuje się odpowiednio plastiki lub żywice[11]. Drukarka SV06

umożliwia druk w technologii FDM, zdecydowano więc na użycie najpopularniejszego plastiku typu PLA *Polylactic Acid*.

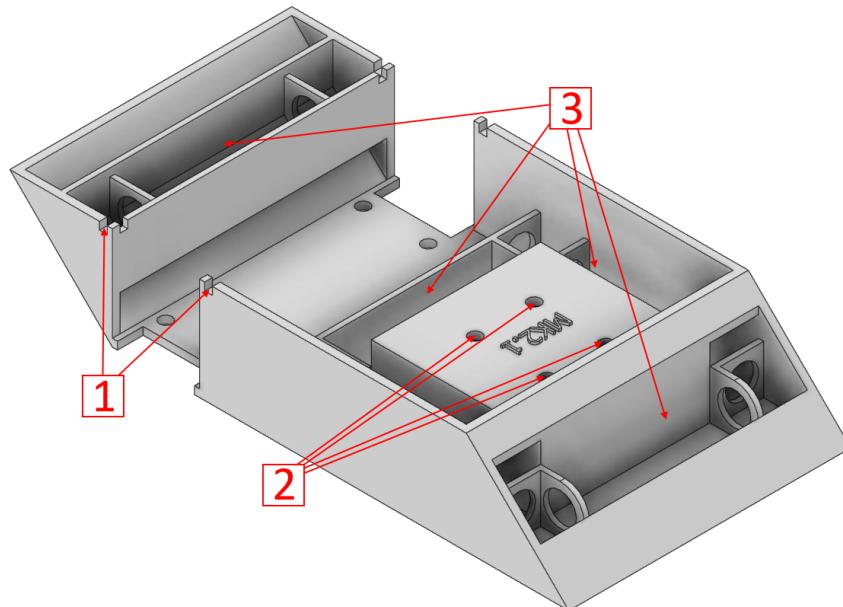
Przed przystąpieniem do projektu, należy rozważyć założenia w kontekście ograniczeń i możliwości zastosowanej technologii. Pierwszym ograniczeniem jest wymiar druku. Główica podająca materiał może w zależności od konfiguracji drukarki poruszać się w różny sposób. 3 podstawowe modele to kartezjański XZ, CoreXY i delta. Pierwszy z wymienionych, a równocześnie najpopularniejszy, wykorzystany został w zastosowanej drukarce SV06. Ruch głowicy polega w nim na przemieszczaniu po standardowych współrzędnych kratezjańskich, x , y , oraz z . Wynika z tego ograniczona wielkość drukowanych modeli — zasięg drukarki jest limitowany. Model SV06 posiada przestrzeń druku w kształcie sześcianu o krawędzi 220 mm. W praktyce, dla bezpieczeństwa drukarki jak i samego druku, odejmuje się pewien margines od każdej krawędzi. Najczęściej jest to około 5 mm z każdej strony w każdym wymiarze, co w przypadku 220 mm ustawia efektywny bezpieczny wymiar druku na 210 mm, dając sześcian o krawędzi 210 mm.

Wiadomo więc, że maksymalny (bezpieczny) rozmiar pojazdu to $210x210x210$ mm. Minimalny wymiar nie jest narzucony odgórnie przez technologię lub założenia, lecz pośrednio przez konieczność ograniczenia masy pojazdu ze względu na skończoną ilość mocy silników. Model pojazdu powinien być więc mniejszy niż maksymalny wymiar, i jednocześnie możliwe jak najmniejszy. Drogą eksperymentalną wyznaczono minimalną grubość ścianek pojazdu, które nie oddają się łatwo zgięciu i zachowują wysoką sztywność, na 3 mm.

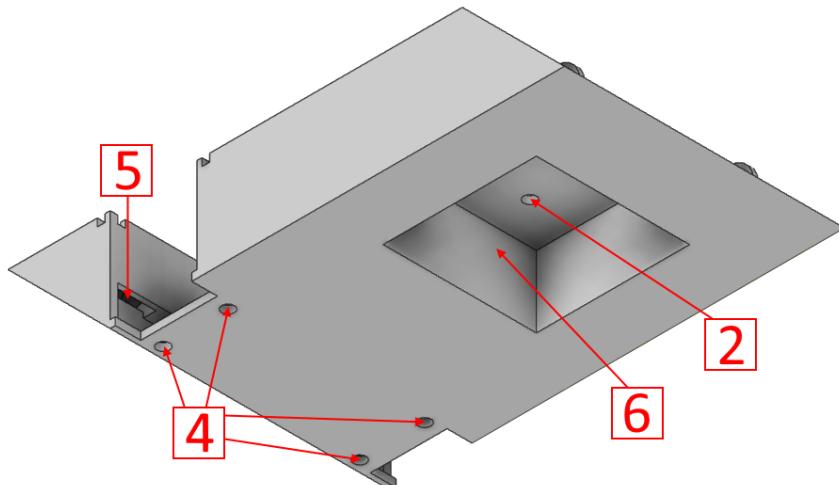
Uwzględnić należy również modułowość pojazdu, wynikającą z założeń. Moduły pojazdu są 2 — pierwszy to sama rama pojazdu, drugi to osłona silników determinująca średnicę kół. Początkowo zakładano stworzenie dodatkowych 2 modułów — pokrycia wierzchniego, oraz przedniej klapy pojazdu. Nie są one wymagane do prawidłowego działania, jednak dodałyby walorów estetycznych ukrywając elementy wewnętrzne. Z tą myślą projektowano model, zostawiając punkty zaczepowe dla odpowiednich modułów. Pomysł został jednak porzucony ze względu na ograniczony czas.

Pierwsza wersja modelu zakładała szerokość 80 mm i długość 120 mm. W trakcie projektowania bardzo szybko stało się jasne, że konieczne będzie pójście na ustępstwa. Pierwszym ustępstwem było porzucenie idei 4 kół. Założenie obiecujące pod kątem wizualnym, lecz w praktyce wymagające zamontowania dodatkowej osi z przodu pojazdu wraz z silnikiem skrętnym, co niepotrzebnie komplikowało projekt. W związku z porzuceniem kół przednich o wspólnej, skrętnej osi, konieczne stało się zastosowanie rozwiązania zastępczego. Wybór padł na pojedyncze obrotowe koło podporowe. Drugim ustępstwem była długość i szerokość. Zastosowanie koła podporowego o średnicy obrotu około 55 mm, w połączeniu z 4 ścianami i 2 akumulatorami zasilającymi, wymusiło zwiększenie szerokości modelu do 110 mm, zaś długości aż do 218 mm, przekraczając tym samym bezpieczną granicę o 8 mm i zbliżając się do limitu możliwości drukarki.

Finalny wygląd modelu 3D ramy przedstawiono na Rysunku 4.1.



(a) Widok z góry



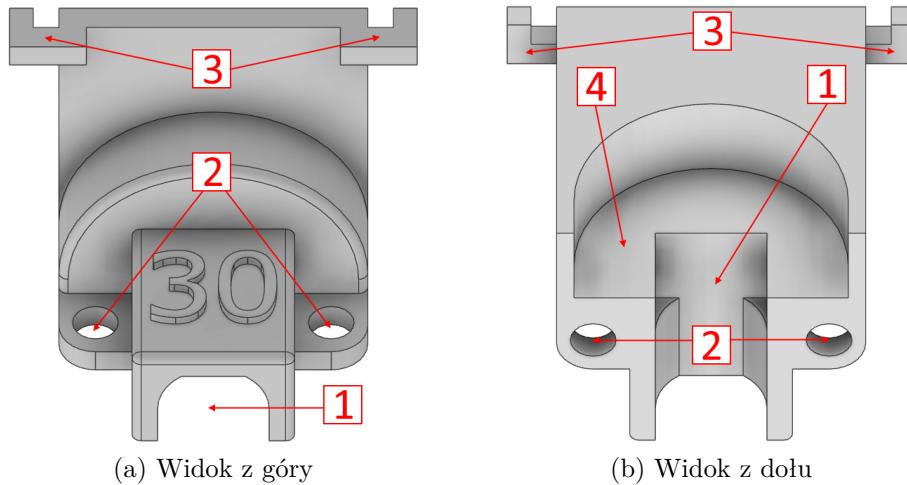
(b) Widok z dołu

Rysunek 4.1: Model 3D ramy pojazdu: a) widok z góry, b) widok z dołu

1. Uchwyty na osłony silników
2. Otwory na śruby mocujące koło podporowe
3. Sloty na akumulatory zasilające
4. Otwory na śruby mocujące osłony silników
5. Otwory na przewody akumulatorów zasilających
6. Przestrzeń na koło podporowe

Drugim modułem modelu 3D są osłony silników. Zakładana odległość pojazdu od ziemi wynosi 5 mm, więc wraz ze wzrostem średnicy kół konieczne jest podniesienie tylnej osi. W przeciwnym wypadku, pojazd zacząłby podnosić się z tyłu, co spowodowałoby przy większych średnicach uderzenie przodu pojazdu o podłożę. Model został więc stworzony

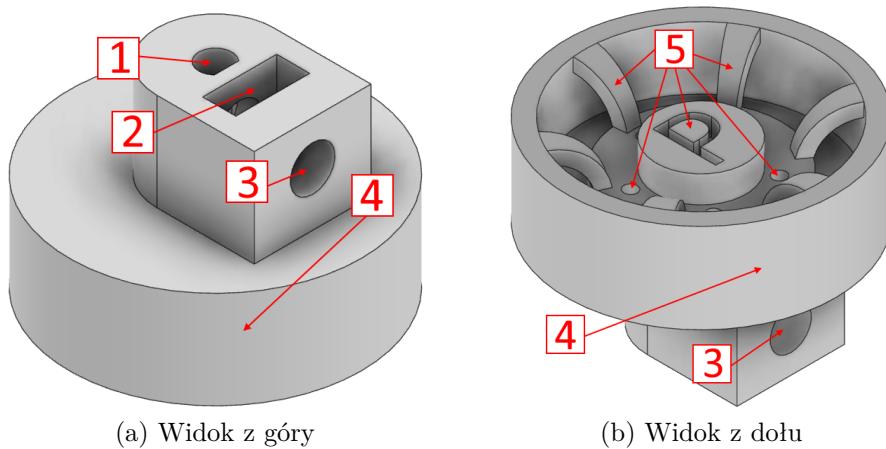
tak, by zmieniać wysokość osi w razie potrzeby. Model osłony silników dla kół o średnicy 30 mm przedstawiono na Rysunku 4.2.



Rysunek 4.2: Model 3D osłony silnika: a) widok z góry, b) widok z dołu

1. Przestrzeń mocowania silnika
 2. Otwory na śruby mocujące osłony silników
 3. Uchwyty przytwierdzające osłony do ramy
 4. Przestrzeń koła

Ostatnim elementem pojazdu są koła. Podstawową wartością w projekcie jest średnica 30 mm. Model widoczny na Rysunku 4.3



Rysunek 4.3: Model 3D osłony silnika: a) widok z góry, b) widok z dołu

1. Otwór wału silnika
 2. Przestrzeń nakładki śruby dociskowej
 3. Przestrzeń śruby dociskowej
 4. Koło
 5. Elementy ozdobne

Elektronika

Podstawową funkcją układu elektronicznego jest obsługa 2 silników DC. Wymaga to zastosowania elementu generującego sygnał sterujący, zasilania, oraz części przekazującej moc do silników. Jako serce układu wybrano mikrokontroler ESP32-DevKitC V4 (Rysunek 4.4). Odpowiada on za wszystkie najważniejsze operacje: odbieranie i wysyłanie danych po Wi-Fi, rozpoczęcie i kończenie pomiarów, oraz generowanie sygnału sterującego. Dokładną specyfikację techniczną pominięto, ponieważ nie jest ona istotna w kontekście projektu. Została dobrana tak, by spełniała założenia projektowe (sterowanie PWM i obsługa Wi-Fi).



Rysunek 4.4: Płytki mikrokontrolerowa ESP32-DevKitC V4

Napędem są 2 silniki DC z enkoderami magnetycznymi widoczne na Rysunku 4.5. Informacyjnie zamieszczono również ich specyfikację w Tabeli 4.1.



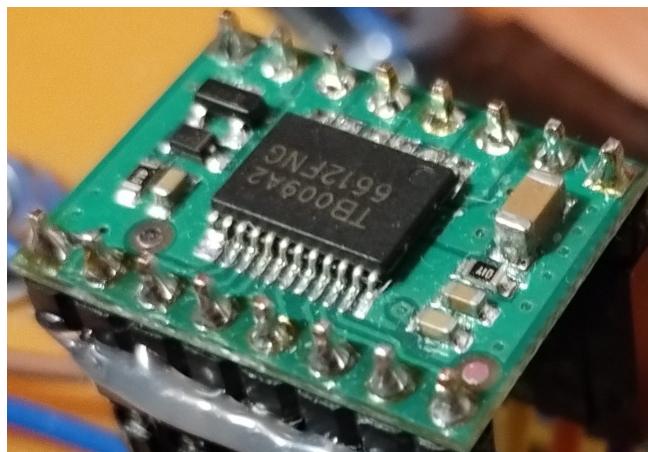
Rysunek 4.5: Silnik DC z enkoderem magnetycznym (Bibliografia: [13])

Tabela 4.1: Specyfikacja silnika DC z enkoderem magnetycznym (Bibliografia: [13])

Specyfikacja silników					
Napięcie znamionowe	12 V	Położenie znamionowe	230 RPM	Położenie na biegu jałowym	300 RPM
Przełożenie	100:1	Moment obrotowy znamionowy	200 g.cm	Moment obrotowy maksymalny	1.6 kg.cm
Prąd blokady	0.9 A	Prąd na biegu jałowym	≤ 75 mA	Prąd znamionowy	≤ 0.3 A
Liczba impulsów enkodera na obrót	7	Napięcie znamionowe enkodera	3.3 V do 5 V	Sygnał wyjściowy enkodera	Cyfrowa funkcja kwadratowa

Do zasilania wykorzystano akumulatory litowo-jonowe typu 18650. Ponieważ pojedynczy akumulator posiada napięcie znamionowe 3.7 V i maksymalne 4.2 V, połączono je szeregowo w 2 grupach po 2 i 3 — dając odpowiednio 6.4 V do 8.2 V i 10.8 V do 12.6 V — w celu zasilania odpowiednio ESP32 i silników.

Zasilania nie można niestety podłączyć bezpośrednio do silników. Spowodowałoby to działanie w trybie ciągłym na pełnej mocy, bez możliwości zmiany polaryzacji, co byłoby całkowicie sprzeczne z założeniami projektowymi. Konieczne jest podłączenie w taki sposób, by możliwe było przekazanie sygnału sterującego z ESP32, oraz zmiana polaryzacji. Zastosowanie połączenia pośredniego przez samą płytę ESP32 jest niestety niemożliwe, ponieważ nie jest w stanie ona obsługiwać tak dużych prądów. Użyto więc w tym celu sterownika silników DC, model TB6612FNG firmy Pololu (Rysunek 4.6). Działa on na zasadzie mostka H, umożliwiającego zmianę polaryzacji. Dzięki budowie dwukanałowej, jeden sterownik umożliwia obsługę 2 silników. Posiada również ochronę przed prądami zwrotnymi, termiczny obwód odcinający i kondensatory filtrujące. Informacyjnie zamieszczono również jego specyfikację w Tabeli 4.1.



Rysunek 4.6: Sterownik silników DC TB6612FNG

Tabela 4.2: Specyfikacja sterownika silników DC TB6612FNG (Bibliografia: [16])

Specyfikacja sterownika TB6612FNG					
Zasilanie silników (VMOT)	4.5 V do 13.5 V	Zasilanie układu logicznego (VCC)	2.7 V do 5.5 V	Maksymalna częstotliwość PWM	100 kHz
Ciągły prąd wyjściowy na kanał	1 A	Maksymalny prąd wyjściowy na kanał	3 A	Łączny ciągły prąd maksymalny	2 A

Dużym wyzwaniem okazał się dobór odpowiedniego czujnika laserowego, który z wysoką precyzją mierzyłby dokładność przebytej odległości i pomiarów z enkoderów. Pierwszym problemem był wybór takiego czujnika, który działałby z odpowiednią precyzją i dokładnością. Wymagana jest dokładność poniżej 5 mm, idealnie w okolicach 1 mm.

Drugi problem to kąt działania czujnika. Większość z nich oferuje kąt działania w okolicach 15° symetryczny wokół środka, dając promień 7.5° . Czujnik znajdowałby się bardzo blisko podłogi, co tworzyłoby ryzyko, że utworzony przez kąt działania stożek uderzałby o podłogę i wykrywał ją jako przeszkodę, lub wykrywał inne niepożądane poboczne obiekty, niwelując tym samym użyteczność otrzymanych pomiarów. Dalmierze oferujące pomiar punktowy lub bliski punktowemu posiadają znaczne wymiary, niską kompatybilność z elektroniką hobbystyczną (ESP32), dużą masę, i/lub wysoką cenę. Istnieją dalmierze hobbystyczne które idealnie nadawałyby się jako element pomiarowy, lecz ich dystans pomiarowy jest niewielki i wynosi kilka do kilkudziesięciu centymetrów. Rozwiązań alternatywnym byłoby wykorzystanie systemu LIDAR (ang. *Light Detection and Ranging*), który umożliwia nie tylko punktowy pomiar odległości, lecz mapowanie danego obszaru wykorzystując technologię czujników laserowych. Istnieją modele spełniające założenia projektowe i mieszczące się w sensownym budżecie, jednak ich wykorzystanie wymagałoby znaczących nakładów czasowych w celu nauki sprzętu, zaprogramowania odpowiednich funkcjonalności, a następnie testów. Z tego powodu zdecydowano o porzuceniu idei pomiaru laserowego i zamiast tego wykorzystanie pomiaru manualnego z pomocą dostępnego powszechnie przymiaru liniowego.

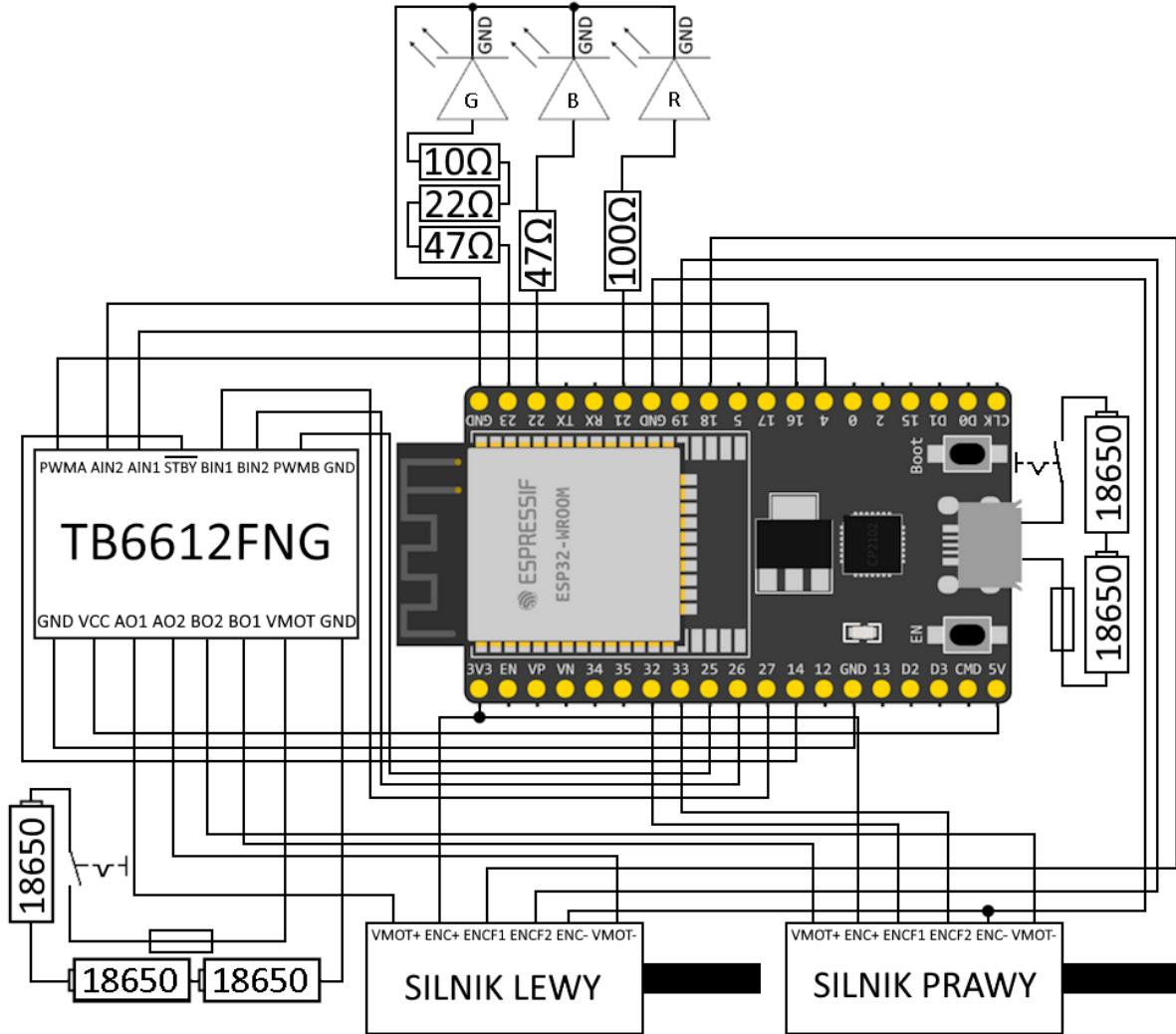
Enkodery magnetyczne zostały zapewnione w silnikach napędowych — stanowią wspólną całość, wymagane jest jedynie ich odpowiednie podłączenie w postaci przewodów zasilających i sygnałowych. Do sygnalizowania stanu oprogramowania ESP32, czyli aktualnego stanu w jakim znajduje się układ, założono instalację LED oraz głośnika. Po zamontowaniu LED w układzie i przetestowaniu rozwiązania, okazało się, że jest ono wystarczające do spełnienia założonego celu. Tym samym, montaż głośnika stał się zbędny i usunięto go z założeń projektowych. Przełączniki źródeł prądowych zamontowano szeregowo przy akumulatorach zasilających, w postaci normalnie otwartych przycisków bistabilnych.

LEDy po podłączeniu standardowego napięcia zasilającego — w tym przypadku zakres od 2 V do 3.3 V, w zależności od koloru diody — doświadczają przepływu prądu znacznie większego, niż prąd nominalny (dla wykorzystanych diod 22 mA). Z tego powodu, w celach ochronnych stosuje się rezystory obniżające wartość przepływającego prądu. W większości przypadków wartość rezystora dobiera się stosując prawo Ohma, dla rzeczywistych układów biorąc również pod uwagę spadek napięcia na diodzie, wynoszący około 0.7 V. W tym projekcie nie ma jednak potrzeby stosowania obliczeń, ponieważ gotowe wartości rezystorów zostały podane przez producenta. 100Ω dla diody czerwonej, 82Ω dla zielonej i 47Ω dla niebieskiej. Z powodu braku dostępności rezystora 82Ω , połączono szeregowo 3 rezystory o wartościach 10Ω , 22Ω i 47Ω , dając łącznie 79Ω . Podłączenie wykonano w układzie wspólnej katody.

Do zabezpieczenia układu wykorzystano popularne bezpieczniki topikowe. Włączone zostały w układ szeregowo przy akumulatorach zasilających. Jako że zarówno sterownik jak i ESP32 posiadają wewnętrzne zabezpieczenia nadprądowe do pewnego progu,

zadaniem bezpieczników nie jest ograniczenie prądu poniżej konkretnych wartości, lecz zabezpieczenie przed zwarciem. Prąd zwarciowy dla akumulatorów typu 18650 wynosi około 30 A do 40 A, czego żaden z układów ani przewodów by nie wytrzymały. Dlatego wartości zostały dobrane arbitralnie i wynoszą 0.1 A i 0.25 A dla 230 V, dając odpowiednio 23 W i 57.5 W. Są to wartości wystarczające by umożliwić układowi swobodne działanie, a jednocześnie zabezpieczyć przed zwarciem.

Pełny schemat układu elektronicznego przedstawiono na Rysunku 4.7



Rysunek 4.7: Schemat układu elektronicznego

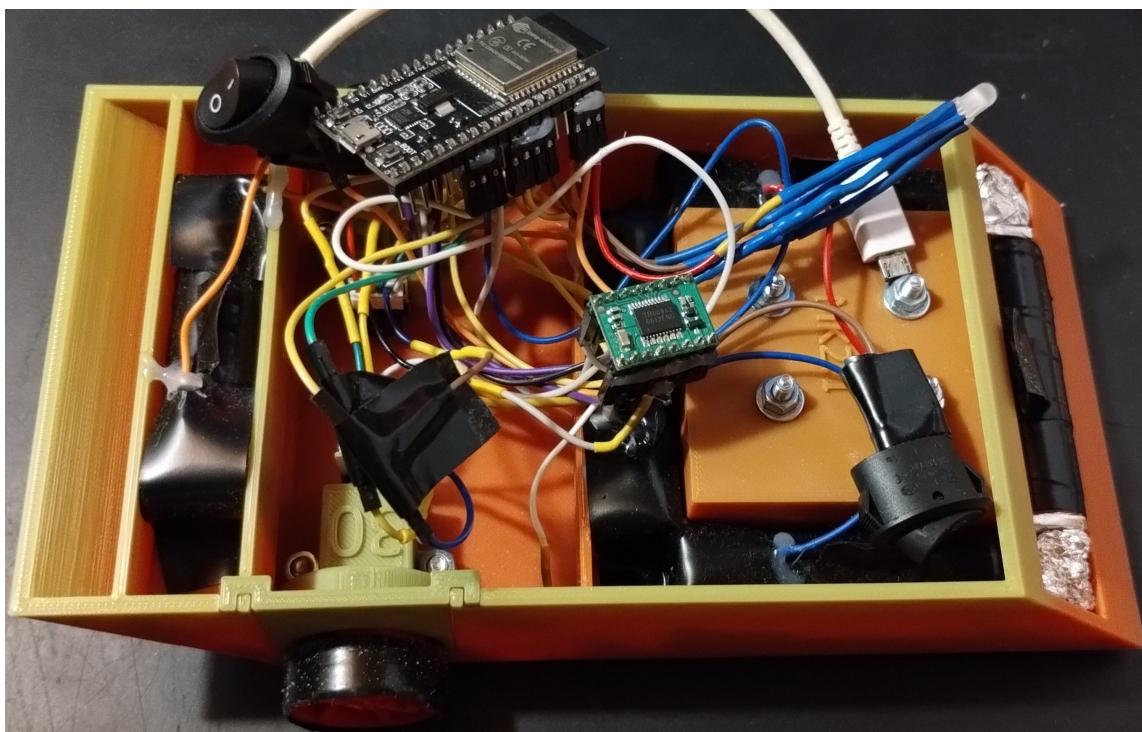
Oznaczenia silników:

1. VMOT+ — zasilanie silników
2. ENC+ — zasilanie enkoderów
3. ENCF1, ENCF2 — sygnały zwrotne enkoderów
4. ENC— — uziemienie enkoderów
5. VMOT- — uziemienie silników

Oznaczenia sterownika:

1. PWMA — wejście sygnału PWM lewego silnika
2. AIN2 — wejście polaryzacji lewego silnika 2
3. AIN1 — wejście polaryzacji lewego silnika 1
4. STBY — tryb postoju/oszczędzania energii (ang. *standby*), aktywacja sygnałem niskim
5. BIN1 — wejście polaryzacji prawego silnika 1
6. BIN2 — wejście polaryzacji prawego silnika 2
7. PWMA — wejście sygnału PWM prawego silnika
8. GND — uziemienie, wszystkie 3 piny są połączone
9. VCC — zasilanie układu logicznego sterownika
10. AO1 — wyjście zasilające lewego silnika 1
11. AO2 — wyjście zasilające lewego silnika 2
12. BO2 — wyjście zasilające prawego silnika 2
13. BO1 — wyjście zasilające prawego silnika 1
14. VMOT — wejście zasilające silników

Na Rysunkach 4.8, 4.9 pokazano końcowy efekt, po złożeniu ze sobą wszystkich komponentów. Niestety w trakcie wykonywania zdjęć, przewód zasilający płytę ESP32 uległ uszkodzeniu przez zmęcenie materiałowe. Jest to uszkodzenie o niskim priorytecie, którego naprawa nie przyniosłaby większych korzyści. Z tego powodu pozostałe badania eksperymentalne i samo nagranie z działania zostały wykonane na zasilaniu ze źródła zewnętrznego, banku energii (ang. *power bank*).



Rysunek 4.8: Zdjęcie pojazdu z góry



Rysunek 4.9: Zdjęcie pojazdu z dołu

Oprogramowanie

Oprogramowanie pojazdu podzielone zostało na kilka modułów, każdy z nich umieszczono w osobnym pliku .cpp. Poszczególne moduły działają na osobnych zadaniach (ang. *task*), w zależności od potrzeb niektóre na więcej niż jednym. ESP32 posiada 2 rdzenie, o numeracji 0 i 1. Rdzeń pierwszy odpowiada domyślnie za obsługę modułów fizycznych, takich jak Wi-Fi i Bluetooth, zaś drugi za wykonanie reszty kodu. Aby zapewnić enkoderom możliwie najwięcej czasu procesora i największą ilość przerwań, moduły odpowiadające za regulator (i tym samym enkodery) umieszczone są na rdzeniu pierwszym, zaś całą resztę na rdzeniu zerowym. Poniżej umieszczone są krótkie opisy poszczególnych modułów, szczegółowe opisy funkcjonalności znajdują się w odpowiedniej dla danego modułu sekcji.

- Moduł główny — zawiera jedynie funkcję startową *app_main*, wywołującą pozostałe moduły (dlatego nie posiada osobnej sekcji)
- Moduł łączności — obsługuje utrzymywane połączenia Wi-Fi, oraz serwera i klienta UDP
- Moduł regulatora — przetwarza脉sy otrzymane z enkoderów, oraz oblicza sygnały sterujące
- Moduł procesora pakietów — odczytuje zawartość ramek z pakietów UDP i zapisuje ich zawartość
- Moduł konfiguracyjny — przechowuje w jednym miejscu wszystkie najważniejsze zmienne konfiguracyjne systemu

Moduł łączności

Po wywołaniu z modułu głównego, moduł łączności rozpoczyna 3 zadania: podtrzymywanie połączenia Wi-Fi, serwera UDP i klienta UDP. Zadanie podtrzymywania Wi-Fi uruchamia połączenie z siecią, a następnie przechodzi do nieskończonej pętli sprawdzającej czy połączenie nie zostało przerwane (Źródło: 1). Jeśli zostało przerwane, podejmowane są odpowiednie czynności — zmiana koloru LED na czerwony, oraz zatrzymanie pomiaru (a co za tym idzie, wyłączenie silników), jeśli był on w trakcie. Wykrywanie przerwania/wznowienia połączenia następuje poprzez obsługę zdarzeń (ang. *event handlers*) platformy (ang. *framework*) ESP-IDF.

Klient UDP tworzy gniazdo (ang. *socket*) z przypisanym adresem IP i portem, a po czym również wchodzi w nieskończoną pętlę wysyłającą pakiety (Źródło: 2). Ponieważ wysyłanie pakietów bez przerwy w nieskończonej pętli byłoby wysoce nieefektywne (na przykład w przypadku braku pakietów do wysłania), dlatego po każdej pętli zadanie jest zatrzymywane i wybudzane z powrotem zewnętrznie przez inne zadanie, gdy pojawi się pakiet do wysłania.

Analogicznie działa zadanie serwera UDP odbierający pakiety, z tą różnicą, że przyjmuje pakiety od dowolnego adresu IP, oraz działa w ciągłej pętli w odstępie zdefiniowanym w pliku konfiguracyjnym (domyślnie 50 ms).

Moduł regulatora

Najważniejszy moduł oprogramowania ESP32. Odczytuje, przetwarza i zapisuje liczbę pulsów enkodera, a następnie na ich podstawie oblicza sygnały sterujące. Obsługa enkoderów została stworzona z wykorzystaniem dodatkowych komponentów platformy otwartoźródłowej ESP-IDF (Bibliografia: [15]).

W pierwszej kolejności wywoływane są funkcje konfiguracyjne pinów, konfigurowane generatory sygnału PWM dla częstotliwości 25 kHz, oraz tworzone zadanie obsługujące pozostałą część kodu. Zadanie to uruchamia następnie zegar cykliczny (co 50 ms) wywołujący funkcję obsługującą odczyt enkoderów oraz wywołującą regulator. Działanie zegara cyklicznego jest zasadniczo identyczne jak nieskończonej pętli, z tą różnicą, że działa na przerwaniach, gwarantując tym samym stały odstęp czasowy między kolejnymi wywołaniami.

Wyrównywanie pozycji silników wykonać można na kilka różnych sposobów. W tym projekcie wykorzystano model "prowadzący-słedzący" (ang. *leader-follower*). W modelu tym, jeden z silników przyjmuje rolę prowadzącego, tym samym posiada w sygnale sterującym tylko jedną, niezależną część, której zadaniem jest doprowadzenie pozycji absolutnej silnika do wartości zadanej. Drugi silnik, będący śledzącym, posiada 2 części sygnału sterującego — niezależną oraz zależną. Część niezależna ma identyczne działanie, jak w przypadku silnika prowadzącego. Zaś część zależna odpowiada za dociągnięcie

błędu między silnikiem prowadzącym a śledzącym do 0. Funkcje silników zostały wybrane arbitralnie przez autora pracy — silnikiem **prowadzącym** jest silnik **prawy**, a silnikiem **śledzącym** jest silnik *lewy*.

Każda z części sygnału sterującego posiada własny, osobno strojony, regulator PID. Silnik prawy posiada 1 część sygnału, silnik lewy 2 części, dając łącznie 3 regulatory PID. W dalszej części pracy, regulatory te będą nazywane dla ułatwienia *lewym*, *prawym*, oraz *synchronizującym*.

Z powodu wysokiej istotności tej części kodu, zostanie ona szczegółowo przeanalizowana i wyjaśniona. Kod jest możliwy do wglądu w Źródło 3. Zaczynając od linijki 90, czyli deklaracji funkcji, widać że przyjmuje ona argument typu *void*. Dodanie argumentu jest wymagane przez zegar cykliczny, jednak nie jest on w żaden sposób wykorzystany. W linijkach 91-92 wywoływana jest funkcja platformy ESP-IDF odczytująca aktualną pozycję absolutną wałów silników względem ich pozycji startowej. Sama pozycja startowa (położenie kątowe) jest nieistotne, ważna jest jedynie odległość od niej. Linijki 93-94 odpowiadają za obliczenie ilości impulsów od ostatniej pętli, a 95-96 zapisują aktualną pozycję absolutną jako poprzednią (względem następnej pętli). Po odczytaniu enkoderów, w linijce 98 wybudzany jest klient UDP w celu wysłania zmierzonych danych.

W liniach 100-107 obliczane są wartości błędów, a następnie sygnałów sterujących. Najpierw dla silnika prowadzącego, następnie dla śledzącego, na końcu synchronizującego. Równanie 4.1 ogólne uchybu (błędu) dla *i*-tej próbki zapisano poniżej.

$$e(i) = w(i) - y(i) \quad (4.1)$$

Dla silnika prowadzącego (linijka 100) wartością zadaną jest położenie absolutne wału silnika wyrażone liczbą pulsów, zaś wyjściem aktualna liczba pulsów, co odpowiada równaniu 4.1. Tak samo wygląda równanie niezależne dla silnika śledzącego, z tą różnicą, że uwzględniony zostaje przeciwny kierunek obrotów silnika. Dzieje się tak, ponieważ silniki są fizycznie obrócone do siebie pod kątem 180° . Zwrot obrotów deklarowany jest binarnie (0 lub 1), silniki będąc tak samo orientowane fizycznie posiadają ten sam zwrot, jednak dla obrotu o 180° należy zmienić deklarowany zwrot obrotów silnika, by uzyskać ten sam zwrot jazdy. Wyjście układu przyjmuje wtedy wartość ujemną, a zmodyfikowane Równanie 4.2 pokazano poniżej. Tylko z pozoru jest to sprzężenie dodatnie, ponieważ wartość *y* jest ujemna.

$$e(i) = w(i) + y(i) \quad (4.2)$$

Kolejna modyfikacja następuje dla sygnału zależnego silnika śledzącego. Wartość zadaną w równaniu 4.2 zostaje zmieniona z położenia absolutnego wału silnika, na aktualną liczbę pulsów silnika prowadzącego, dociągając błąd między położeniem silnika śledzącego a prowadzącego do 0.

Po obliczeniu błędu, należy obliczyć wartość sygnału sterującego regulatorami PID. Równanie 4.3 przedstawia ogólną postać dyskretną regulatora PID. W Źródło 4 (Bibliografia: [15][8]) wykorzystano postać niezależną regulatora PID (Równanie 4.4).

$$u(i) = u_P(i) + u_I(i) + u_D i \quad (4.3)$$

$$u(i) = k_P e(i) + k_I \sum_{j=0}^i e(j) + k_D (e(i) - e(i-1)) \quad (4.4)$$

W linii 73 (Źródło 4) sumowany jest uchyb. Aby wyeliminować zjawisko nawijania błędu (ang. *windup*) spowodowane sumowaniem błędu w kolejnych próbkach do zbyt dużych wartości, co może powodować dominację sygnału całkującego nad proporcjonalnym i różniczkującym, a także opóźnić odpowiedź na zmianę wartości zadanej, zastosowano mechanizm zapobiegający (ang. *anti-windup*) w postaci ograniczenia całkowania. Przedział został wyznaczony eksperymentalnie wspomagając się charakterystyką statyczną (Rysunek ??) i wygląda następująco: $EI \in \langle -100; 100 \rangle$, gdzie EI to suma błędu (ang. *Error Integral*). Ograniczenie widoczne jest w liniach 74-75.

Następnie wyjście z regulatora ograniczane jest do zakresu $u \in \langle -u_{\max}; u_{\max} \rangle$. u_{\max} wyznaczono na podstawie charakterystyki statycznej (Rysunek ??). Widoczne jest na niej, że maksymalną wartością dla jednej pętli dla pełnej mocy jest liczba w przedziale $\langle 500; 600 \rangle$, eksperymentalnie wyznaczono jej wartość na $p_{\max} = 571$. Biorąc pod uwagę dodanie obciążenia z lekkim zapasem, wyznaczono maksymalną dozwoloną liczbę pulsów na pętlę na 400. W pliku konfiguracyjnym (Config.cpp), zapisano limit w postaci $u_{\max} = 100\alpha$, gdzie α to współczynnik w przedziale $\langle 1; \frac{p_{\max}}{100} \rangle$. Technicznie możliwa jest wartość α również w przedziale $\langle 0; 1 \rangle$, jednak nie ma ona sensu z inżynierskiego punktu widzenia — silnik będzie obracał na bardzo niskich obrotach, a po dołożeniu obciążenia istnieje ryzyko, że nie pokona oporu statycznego, tym samym nie ruszając z miejsca.

Na końcu (linijka 85), aktualna wartość błędu zapisywana jest do obliczenia składowej różniczkowej sygnału w następnej pętli, a aktualna wartość sygnału zwracana przez funkcję (linijka 87).

Wracając do kodu pętli (Źródło 3), po otrzymaniu wartości sygnału ze wszystkich 3 regulatorów PID, następuje jego przetworzenie przed wysłaniem na sterownik. W liniach 109-110 składowa zależna i niezależna sygnału sterującego silnika śledzącego są sumowane, a następnie ich suma ponownie ograniczana do u_{\max} .

W następnym kroku (linie 115-116), otrzymany wynik zamieniany jest na wartość procentową (Równanie 4.5), ponieważ taką przyjmuje jako argument wykorzystany generator sygnału PWM.

$$u(i) = u(i)(100/p_{\max}) \quad (4.5)$$

Ostatnim krokiem przetworzenia sygnału sterującego jest ograniczenie go do wartości zadeklarowanej przez użytkownika w aplikacji (linie 115-116), mnożąc go przez pożądaną wartość. Na przykład, dla 70% mocy natępuje mnożenie razy 0.7. Aby zapewnić silnikowi śledzącemu odrobinę zapasu mocy w razie potrzeby, moc silnika prowadzącego jest dodatkowo zmniejszana o 10%.

Przed wysłaniem pożądanej wartości do generatora PWM, konieczne jest określenie zwrotu obrotu silników. Dzieje się to w liniach 118-128 na podstawie znaku aktualnej wartości błędu. Po tej czynności, wartość sygnału zostaje wysłana do generatora PWM (linie 130-131), który generuje sygnał i wysyła go na sterownik.

Moduł procesora pakietów

Jest to moduł zawierający 3 funkcje — przetwarzającą ramkę, sprawdzającą klucz i sprawdzającą wartość. Funkcja przetwarzająca ramkę jest wywoływana z modułu łączności po każdym odebranym pakiectwie UDP. Funkcja ta na początku wywołuje funkcję sprawdzającą klucz. Jest to prosty mechanizm zabezpieczający przed odczytywaniem niepotrzebnych pakietów. Jako że ESP32 może otrzymywać dowolne pakiety z dowolnego pakietu, w trakcie testów kilka razy zdarzyło się, że funkcja otrzymywała wiadomości inne niż pożądane, z którymi sobie nie radziła. Jeśli ramka nie zawiera klucza, funkcja kończy działanie. Zmiana klucza możliwa jest w pliku konfiguracyjnym, jednak trzeba pamiętać o zmianie jego długości w odpowiedniej pętli odczytującej. Ponieważ nie została zaimplementowana żadna metoda szyfrowania, zmiana klucza ramki w żaden sposób nie zwiększa cyberbezpieczeństwa. Służy on jedynie do identyfikacji źródła wiadomości.

Jeśli zawiera klucz, rozpoczyna odczytywanie danych z ramki. Ma ona postać *KLU-CZA1B2C3D4Z*. Każda litera następująca po kluczu oznacza inną zmienną. Dla przykładu *A* oznacza wartość zadaną, *B* wzmacnienie K_P regulatora PID silnika śledzącego, itp. W momencie pisania tego tekstu, ramka sięga litery *K*. Po wykryciu litery, ramka przekazywana jest do funkcji odczytującej następującą po niej wartość. Po zwróceniu wartości, zapisywana jest ona do odpowiedniej zmiennej. Proces ten trwa do napotkania litery *Z*, oznaczającej koniec ramki. Osobnym oznaczeniem jest literka *Q* oznaczająca wyście (ang. *quit*), która nie posiada żadnej wartości, lecz informuje program o zakończeniu pomiaru.

Moduł konfiguracyjny

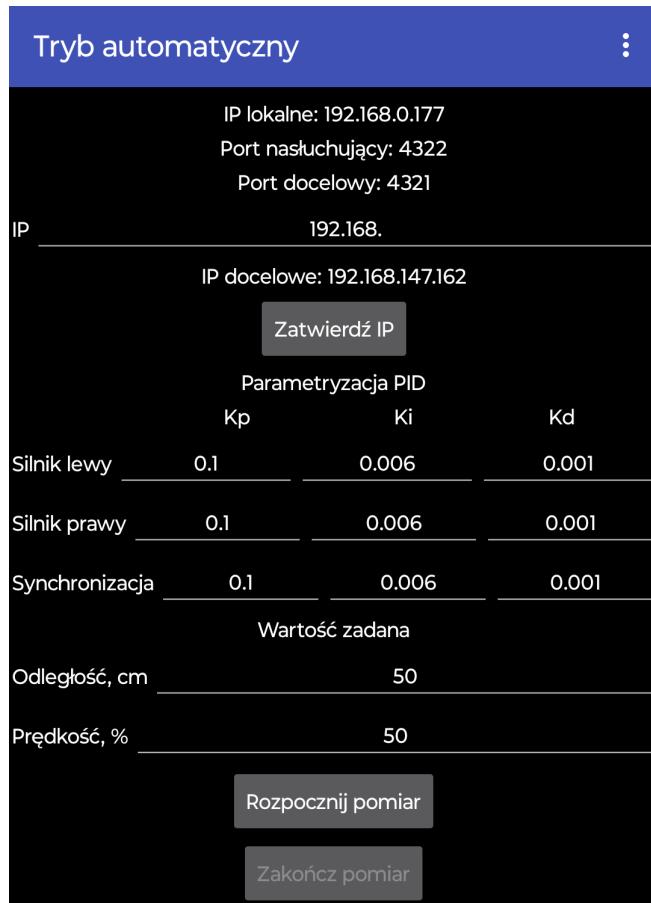
Jego istnienie nie jest technicznie wymagane do działania programu, lecz znacząco zwiększa komfort użytkowania. Zawiera najważniejsze zmienne i instancje struktur programu, tak, by możliwa była wygodna zmiana ich wartości w dowolnym momencie, bez konieczności zmiany wszystkich wystąpień w kodzie. Zmniejsza to również ryzyko pomyłek przez przypadkowe pominięcie wystąpienia danej zmiennej.

4.2 Oprogramowanie pomocnicze

Do wykonania projektu koniecznych było kilka dodatkowych narzędzi pomocniczych.

Aplikacja

Głównym z narzędzi jest aplikacja na system Android, przesyłająca dane użytkownika do systemu. Ponieważ została utworzona w internetowym kreatorze, autorowi nie jest znany jej dokładny kod. Paczka plików aplikacji przed komplikacją jest dołączona do pracy jako załącznik. Na Rysunku 4.10 pokazano wygląd aplikacji po uruchomieniu, z domyślnymi wartościami.



Rysunek 4.10: Zrzut ekranu aplikacji sterującej

Skrypt Python

Krótki skrypt działający w pętli — na początku tworzy pusty plik .csv, następnie oczekuje na pomiar. Po otrzymaniu danych pomiarowych i zapisaniu ich do utworzonego pliku .csv, oczekuje na interakcję użytkownika. W tym momencie można skopiować otrzymane dane poprzez skopiowanie pliku lub zmianę jego nazwy. Na komendę użytkownika, skrypt czyści plik .csv i pętla zaczyna się od nowa. Wadą skryptu jest brak możliwości zakończenia go innej, niż zamknięcie procesu.

Skrypt MATLAB

Mimo nazwy podrozdziału, skrypty w MATLABie są ostatecznie dwa, nie jeden, choć pełnią podobne funkcje. Rozdzielone zostały dla czystości i wygody użytkowania. Pierwszy z nich wykresla charakterystykę statyczną, będącą funkcją zależności pulsów na pętlę od prędkości w biegu jałowym. Szczegółowy opis znajduje się w Rozdziale 5. Robi to odczytując pomiary z zakresu prędkości $\langle 0; 100 \rangle$, ze skokiem równym 5%, a następnie na ich podstawie rysując i łącząc punkty.

Drugi skrypt wykresla 3 wykresy. Pierwszym jest pozycja silników, wyrażona w funkcji zależności pozycji silników od czasu. Drugi pokazuje zadaną w danym momencie przez regulatory PID wartość prędkości zadanej w procentach. Innymi słowy, jest to wykres zależności prędkości zadanej (w procentach) silników od czasu. Trzeci przedstawia prędkość rzeczywistą silników w postaci funkcji zależności rzeczywistej prędkości silników (w pulsach na pętlę) od czasu. Na każdym wykresie ukazane są 2 funkcje, po jednej na silnik.

Rozdział 5

Eksperyment

Work in progress.

Rozdział 6

Podsumowanie i wnioski

Work in progress.

Bibliografia

- [1] Raymond A. DeCarlo i Stanisław H. Żak. „A Quick Introduction to Sliding Mode Control and Its Applications 1”. W: 2008. URL: <https://api.semanticscholar.org/CorpusID:16461642> (term. wiz. 27.01.2024).
- [2] Wielu edytorów. *Ctesibius Of Alexandria*. URL: <https://www.britannica.com/biography/Ctesibius-of-Alexandria> (term. wiz. 29.12.2023).
- [3] Michael Faraday. „On some new Electro-Magnetical Motions, and on the Theory of Magnetism.” W: *The Quarterly Journal of Science, Literature and The Arts* 12 (1822), s. 74–96.
- [4] jjbeard. *A Rotary Encoder Disc with a 3-Bit Binary Reflected Gray Code (BRGC)*. URL: [https://en.wikipedia.org/wiki/Rotary_encoder#/media/File:Encoder_Disc_\(3-Bit\).svg](https://en.wikipedia.org/wiki/Rotary_encoder#/media/File:Encoder_Disc_(3-Bit).svg) (term. wiz. 27.12.2023).
- [5] Nicolas Minorsky. „DIRECTIONAL STABILITY OF AUTOMATICALLY STEERED BODIES”. W: *American Society of Naval Engineers* 34 (1922). ISSN: 0028-1425. DOI: 10.1111/j.1559-3584.1922.tb04958.x.
- [6] Piotr Mitros. *Optical Encoder Project*. URL: <https://groups.csail.mit.edu/mac/users/pmitros/encoder/> (term. wiz. 28.12.2023).
- [7] University of Hawaii NASA Infrared Telescope Facility (IRTF) Institute for Astronomy. *Techniques For Digitizing Rotary and Linear Motion*. URL: http://irtfweb.ifa.hawaii.edu/~tcs3/tcs3/0306_conceptual_design/Docs/05_Encoders/encoder_primer.pdf (term. wiz. 27.12.2023).
- [8] Paweł Nowak. *Cyfrowy regulator ciągły PID*. Instrukcja Laboratorium Urządzeń Automatyki. 2023.
- [9] Alexander S. Poznyak i Yury V. Orlov. „Vadim I. Utkin and sliding mode control”. W: *Journal of the Franklin Institute* 360.17 (2023), s. 12892–12921. ISSN: 0016-0032. DOI: 10.1016/j.jfranklin.2023.09.028.
- [10] Ruchika i Neha Raghu. „Model Predictive Control: History and Development”. W: *International Journal of Engineering Trends and Technology (IJETT)* 4 (2013). ISSN: 2231-5381.

- [11] Krzysztof Serafin. „Bezprzewodowe sterowanie manipulatorem z wykorzystaniem Arduino”. W: (2023), s. 9–14.
- [12] Jonathan Seybold, André Bülau, Karl-Peter Fritz, Alexander Frank, Cor Scherjon, Joachim Burghartz i André Zimmermann. „Miniaturized Optical Encoder with Micro Structured Encoder Disc”. W: *Applied Sciences* 9 (2019). DOI: 10.3390/app9030452.
- [13] JacksonA Tools Store i F***s. URL: <https://pl.aliexpress.com/item/1005004999529855.html> (term. wiz. 29.01.2024).
- [14] *SV06 User Manual V1.0*. SOVOL TECHNOLOGY CO., LIMITED. Rm 23 9 F Blk G Kwai Shing Ind Bldg Stage 2 42 46 Tai Lin Pai Rd Kwai Chung NT 999077 Hong Kong (SAR), 2022.
- [15] Espressif Systems. *Espressif IDF Extra Components*. <https://github.com/espressif/idf-extra-components/tree/master>. 2016-2024.
- [16] Toshiba. URL: https://www.pololu.com/file/download/TB6612FNG.pdf?file_id=0J86 (term. wiz. 29.01.2024).
- [17] C. F. Winder. „Shaft Angle Encoders”. W: *Electronic Industries* 18.10 (1959), s. 76–80.
- [18] Politechnika Śląska. *Wymagania do pracy inżynierskiej dla kierunku Automatyka i Robotyka*. 2021. URL: https://www.polsl.pl/rau/wp-content/uploads/sites/42/2021/11/AiR_Wymagania_do_pracy_inzynierskiej.pdf (term. wiz. 27.12.2023).

Dodatki

Spis skrótów i symboli

PID regulator Proporcjonalno-Całkującco-Różniczkujący (ang. *Proportional–Integral–Derivative*)

MPC regulator oparty na modelu predykcyjnym (ang. *Model Predictive Control*)

DC prąd stały (ang. *Direct Current*)

LED dioda emitująca światło (ang. *Light Emitting Diode*)

3D 3 wymiary (ang. *3 dimensions*)

Wi-Fi sieć bezprzewodowa (ang. *Wireless Fidelity*)

RTOS system czasu rzeczywistego (ang. *Real Time Operating System*)

FDM modelowanie przez nakładanie stopionego materiału (ang. *Fused Deposition Modeling*)

DLP modelowanie przez utwardzanie materiału światłem (ang. *Digital Light Processing*)

PLA filament do druku 3D, polilaktyd (ang. *Polylactic Acid*)

LIDAR technologia mapowania przestrzeni 3D bazująca na detekcji światła i pomiarze odległości (ang. *LIDAR*)

u sygnał sterujący obiektem wykonawczym w układzie sterowania/regulacji

$u(i)$ sygnał sterujący w i -tej chwili czasu (pętli)

u_{\max} maksymalna wartość sygnału sterującego

EI suma błędu (ang. *Error Integral*)

p_{\max} maksymalna liczba pulsów silnika na pętlę na biegu jałowym, wartość stała

α współczynnik maksymalnej prędkości silników, w przedziale $\left\langle 1; \frac{p_{\max}}{100} \right\rangle$

Źródła

Źródło 1: Connectivity.cpp, pętla *while(true)* zadania podtrzymującego połączenie Wi-Fi

```
141     while(true){  
142         if(!connected){  
143             ESP_LOGW("Wi-Fi", "Brak polaczenia z wifi.  
Restartowanie modulu wifi i ponowne laczenie.");  
144             esp_wifi_stop();  
145             vTaskDelay(1000 / portTICK_PERIOD_MS);  
146             esp_wifi_start();  
147             esp_wifi_connect();  
148             while(!connected) vTaskDelayUntil(&xLastWakeTime,  
wifiServiceCheckConnectionConfig.taskSecondLoopDelay /  
portTICK_PERIOD_MS);  
149         }  
150         vTaskDelayUntil(&xLastWakeTime,  
wifiServiceCheckConnectionConfig.taskLoopDelay /  
portTICK_PERIOD_MS);  
151     }
```

Źródło 2: Connectivity.cpp, pętla *while(true)* zadania klienta UDP

```
214     while (1) {  
215         ulTaskNotifyTake(pdTRUE , portMAX_DELAY);  
216         const char *message = udpPreparePayload();  
217         int err = sendto(sock, message, strlen(message), 0, (br/>218             struct sockaddr *)&dest_addr, sizeof(dest_addr));  
219         if(errno == 12){  
220             ESP_LOGW("UDP Client", "Urzadzenie docelowe jest  
offline, nie mozna wyslac pakietu.");  
221         } else if(err < 0){  
222             ESP_LOGE("UDP Client", "Blad przy wysylaniu pakietu,  
nr bledu: %d", errno);  
223         } else {  
224             ESP_LOGI("UDP Client", "Wyslano pakiet danych.");  
225         }  
226     }
```

Źródło 3: Controller.cpp, funkcja wywoływana przez zegar cykliczny

```
90 static void motorLoop(void *args){  
91     pcnt_unit_get_count(leftMotorProperties.pcntUnit, &  
92         leftMotorProperties.pulses);  
93     pcnt_unit_get_count(rightMotorProperties.pcntUnit, &  
94         rightMotorProperties.pulses);  
95     leftMotorProperties.loopPulses = leftMotorProperties.pulses -  
96         lastPulseCountLeftMotor;  
97     rightMotorProperties.loopPulses = rightMotorProperties.pulses -  
98         lastPulseCountRightMotor;  
99     lastPulseCountLeftMotor = leftMotorProperties.pulses;  
100    lastPulseCountRightMotor = rightMotorProperties.pulses;  
101  
102    xTaskNotifyGive(udpClientConfig.taskHandle);  
103  
104    float errorDistanceRight = rightMotorPid.setPoint -  
105        rightMotorProperties.pulses;  
106    float motorPowerRight = getPIDResult(&rightMotorPid,  
107        errorDistanceRight);  
108  
109    float errorDistanceLeft = rightMotorPid.setPoint - (-  
110        leftMotorProperties.pulses);  
111    float motorPowerLeft = getPIDResult(&leftMotorPid,  
112        errorDistanceLeft);  
113  
114    float errorSync = rightMotorProperties.pulses - (-  
115        leftMotorProperties.pulses);  
116    float motorPowerSync = getPIDResult(&leftMotorSyncPid,  
117        errorSync);  
118  
119    motorPowerLeft += motorPowerSync;  
120    motorPowerLeft = MIN(motorPowerLeft, pulsesPerPowerPercent *  
121        100);  
122  
123    motorPowerRight = motorPowerRight * (pulsesPerPowerPercent *  
124        100.0 / maxPulsesPerPowerPercent) / pulsesPerPowerPercent;  
125    motorPowerLeft = motorPowerLeft * (pulsesPerPowerPercent *  
126        100.0 / maxPulsesPerPowerPercent) / pulsesPerPowerPercent;  
127  
128    rightMotorProperties.motorSpeed = motorPowerRight * 0.01 *  
129        maxMotorSpeed * 0.9;
```

```

116     leftMotorProperties.motorSpeed = motorPowerLeft * 0.01 *
117         maxMotorSpeed; /* (pulsesPerPowerPercent * 100 / 600);
118
119     if(errorDistanceRight >= 0){
120         rightMotorProperties.motorDir = 1;
121     } else {
122         rightMotorProperties.motorDir = 0;
123     }
124
125     if(errorDistanceLeft >= 0){
126         leftMotorProperties.motorDir = 0;
127     } else {
128         leftMotorProperties.motorDir = 1;
129     }
130
131     driveMotor(&rightMotorProperties);
132     driveMotor(&leftMotorProperties);
133 }
```

Źródło 4: Controller.cpp, funkcja regulatora wywoływana wewnętrz funkcji wywoywanej przez zegar cykliczny

```

70 static float getPIDResult(pidConfig *pid, float error)
71 {
72     float output = 0;
73     integral_err += error;
74     integral_err = MIN(integral_err, pid->params.max_integral);
75     integral_err = MAX(integral_err, pid->params.min_integral);
76
77     // u(i) = Kp*e(i) + Ki*integral_err + Kd*(e(i)-e(i-1))
78     output = error * pid->params.kp +
79             (error - previous_err1) * pid->params.kd +
80             integral_err * pid->params.ki;
81
82     output = MIN(output, pid->params.max_output);
83     output = MAX(output, pid->params.min_output);
84
85     previous_err1 = error;
86
87     return output;
88 }
```


Lista dodatkowych plików, uzupełniających tekst pracy

- Kod mikrokontrolera ESP32
- Kod skryptu MATLAB
- Kod skryptu Python
- Pliki projektu aplikacji Android
- Dane pomiarowe
- Film prezentujący działanie

Spis rysunków

2.1	Poglądowy schemat dysku enkodera absolutnego z 3-bitowym kodem Graya (Bibliografia: [4])	4
2.2	Poglądowe schematy dysku enkodera inkrementalnego (Bibliografia: [6])	5
2.3	Zobrazowanie zasady działania enkodera magnetycznego dla tarczy z jednym rzędem (Bibliografia: [12])	5
2.4	Zobrazowanie zasady działania regulatora Ktesibiosa	6
2.5	Genealogia algorytmów MPC (Bibliografia: [10])	7
3.1	Drukarka 3D model Sovol SV06 (Bibliografia: [14])	11
4.1	Model 3D ramy pojazdu: a) widok z góry, b) widok z dołu	15
4.2	Model 3D osłony silnika: a) widok z góry, b) widok z dołu	16
4.3	Model 3D osłony silnika: a) widok z góry, b) widok z dołu	16
4.4	Płytnica mikrokontrolerowa ESP32-DevKitC V4	17
4.5	Silnik DC z enkoderem magnetycznym (Bibliografia: [13])	17
4.6	Sterownik silników DC TB6612FNG	18
4.7	Schemat układu elektronicznego	20
4.8	Zdjęcie pojazdu z góry	21
4.9	Zdjęcie pojazdu z dołu	22
4.10	Zrzut ekranu aplikacji sterującej	27

Spis tabel

4.1 Specyfikacja silnika DC z enkoderem magnetycznym (Bibliografia: [13]) . . .	17
4.2 Specyfikacja sterownika silników DC TB6612FNG (Bibliografia: [16]) . . .	18